ENERGY CONSUMPTION-THRESHOLD MONITORING SYSTEM

Web User Interface

Lukáš Kozák

Bachelor's Thesis
May 2011

Degree Programme in Information Technology
School of Technology

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Author(s)

KOZÁK, Lukáš

Type of publication
Bachelor´s Thesis

Date
May 9, 2011

Pages
61

Language
English

Confidential

( ) Until

Permission for web publication
(Yes)

Title
ENERGY CONSUMPTION-THRESHOLD MONITORING SYSTEM
Case: Web User Interface

Degree Programme

Information Technology

Tutor

MIESKOLAINEN, Matti

Assigned by

Energiakolmio Oy

Abstract

There is a general need for as small energy consumption as possible and for a good overview of the consumed energies. This can be achieved by careful and continuous monitoring of deviations in the energy consumption and evaluating the results in the real time.

Energiakolmio has a sophisticated energy reporting system that can be improved with such a service. The service fires so called alarms – warnings about consumption threshold overruns. The user interface of this service is a web solution. The objective of this thesis is to describe, design, implement, and evaluate this web solution.

The specification contains information about types of alarms, messages that are sent to the users, functions that the system performs and a proposal of the user interface.

The web solution is implemented in Microsoft .NET platform, using the Model-View-Presenter software development pattern and Telerik RadControls. Web development related issues are discussed and the prototype is presented. After the implementation and testing phases it is necessary to extract the experience from the project and enclose it with an evaluation.

Regarding to the product owner and team management the project ended up very well. The design of the system and used components were suitable for this solution and the company's knowledge and software bases were enhanced with new elements.

Keywords
efficient energy consumption, consumption threshold overrun, model-view-presenter, Telerik, web

Miscellaneous
This is a version for online publication. Due to fact that parts of the thesis are confidential most figures are omitted in the online publication.

CONTENT

TERMS

ASP	–	Active Server Pages - a web application framework by Microsoft
CVS	–	Concurrent Versions System - a source code version control, allows developers to collaborate
HTML	–	Hypertext Markup Language - markup language for web pages
HTTP	–	Hypertext Transfer Protocol - a networking protocol for data communication in the World Wide Web
EF	–	Entity Framework - object-relational mapping framework for the .NET platform
MS	–	Microsoft - the owner of the .NET platform
MSDN	–	Microsoft Developer Network - enables communication between developers and Microsoft
MVP	–	Model-View-Presenter - software pattern mostly used for building user interfaces
ORM	–	Object-relational Mapping - a technique for converting data
SVN	–	Subversion - a source code version control, allows developers to collaborate
SQL	–	Structured Query Language - a language for communication with databases
TFS2010	–	Team Foundation Server 2010 - a Microsoft's product for source code version control and more
UI	–	User Interface - the place for interaction between a human and a machine
XML	–	Extensible Markup Language - a language for encoding documents

LIST OF TABLES

LIST OF FIGURES

LISTINGS

1 INTRODUCTION AND OBJECTIVE OF THE THESIS

The objective of this thesis was to build a web user interface of a system for needs of an energy company. In order to understand the demand for such a system, one needs to have a basic idea about the energy business. This is discussed in the first chapters. Next, the motivation for this project is presented, followed by incorporation into the company's existing system.

1.1 Energiakolmio

Energiakolmio is an independent organization providing expert services to actors on the energy market and outsourced solutions for energy management. Their know-how combines all the most important areas of the energy sector that contribute to profit-making. By adapting expertise into solutions needed by their customers they are able to provide them with exceptional and unique added value.

Energiakolmio's customers are large and middle-sized companies and public sector organizations in all fields of business. The company's relationship with customers is that of a partner in cooperation - they are offered experience, best practices and tools for their individual needs. The processes and applications in use belong to Energiakolmio. The company's own IT service production enables constant product development and pace setting in the field.

The firm has achieved the leading position in Finland by responding with quality to the needs and challenges created by the changing energy market. It is located in Jyväskylä, Finland, but currently the owner is Alpiq, a corporation with origins in Switzerland. Over 60 professionals are employed by the company at the time being.

Energiakolmio also offers services in the areas of energy acquisition, energy efficiency, invoice logistics and IT services. The service selection has extended to include comprehensive energy management to which *EnerKey.com* web concept offers an outsourced solution. Cooperation with the company releases the customers' own resources to concentrate on their core business.

This and more information can be found on the following website:
`http://www.energiakolmio.fi`

1.2 Energy business

Figure 1 displays how consumption reading data is transferred from a facility to company's database. Each facility has one or more meters which are connected to modems. Devices read consumption information from meters (usually) every hour. Modems collect this piece of information, an XML file is created and a connection to company's server with a parsing tool is made.

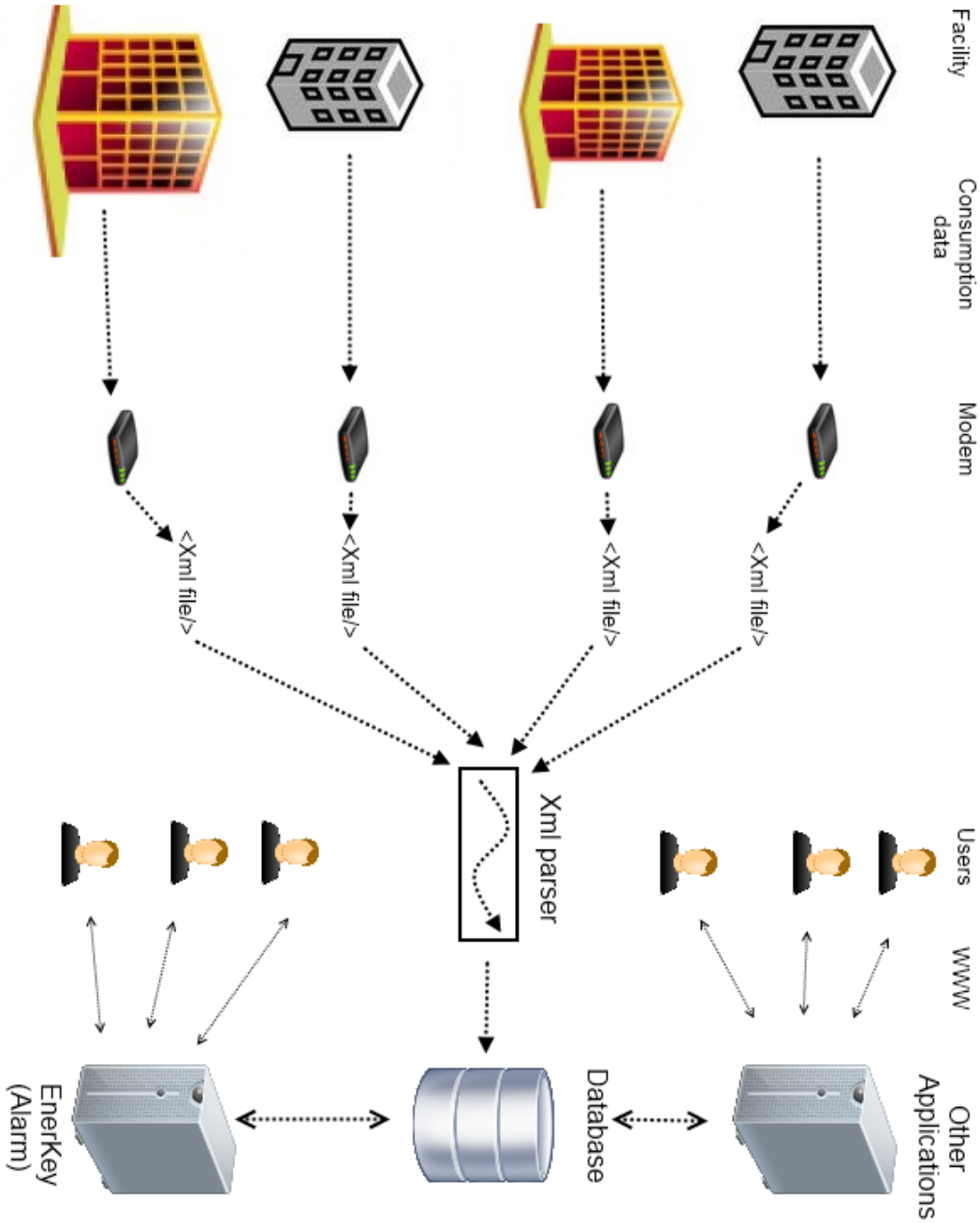

FIGURE 1: Common remote consumption reading architecture

Next, the data is processed by the parser and uploaded to a database which serves as a source of information for all Energiakolmio's applications, including Alarm. This database is replicated for backups, tests and developing purposes. EnerKey Internet service and used energy calculation and remote reading solutions are the company's own products.

An efficient metering and follow-up system is the key to efficient energy consumption as well as to continuous utilization of energy markets. With automatic remote reading, the customers receive up-to-date information of energy consumption by estate or facility. The follow-up and reporting system EnerKey operates via the Internet; the information is available to customers anywhere and anytime.

1.3 Motivation for developing Alarm

Accurate hourly and up-to-date information as the basis of energy consumption allows customers to improve consumption habits. By quickly detecting deviations in consumption, **significant cost savings** can be achieved.

In addition to the above, reporting can also be combined with expert services that enable identification and utilization of potential cost savings.

Addressing deviations in consumption is an interest of company's customer, therefore it is an interest of the company. Of course it is not humanly possible to monitor every single energy meter twenty four hours per day, seven days per week. Energiakolmio already has automatic measurement of energy consumption. By adding functionality, with technology available at the state of the art, it is certainly possible to develop an automated system which could **monitor deviations for humans** and inform about them.

The objective of this thesis is to build such a system, explain its behavior, describe the development phase, present the final result and evaluate used components and the system itself.

1.4 EnerKey and Alarm

EnerKey web service consists of tens of applications with versatile purposes, mostly for monitoring energy flows. These applications are grouped into three categories: EnerControl, EnerPro and EnerCount. Subject of this thesis is energy consumption-threshold monitoring system further mentioned as Alarm, a part of EnerControl.

As can be observed in figure 2, many applications are built to operate with collected energy readings. The company has reports directly working with the readings (yearly reports, monthly reports, hourly reports, benchmarks), followed by applications specialized in transferring energy readings into financial data (cost report, budget creation, budget history), accompanied with energy certificate application, the alarm service and readings oriented services (missing readings, manual reading input, reading correction, facility information). As Energiakolmio has many customers, more holistic views over the energy markets are useful. For this purpose there are Market review, Market info and Emission trading applications. For some customers Invoicing services are provided as well.

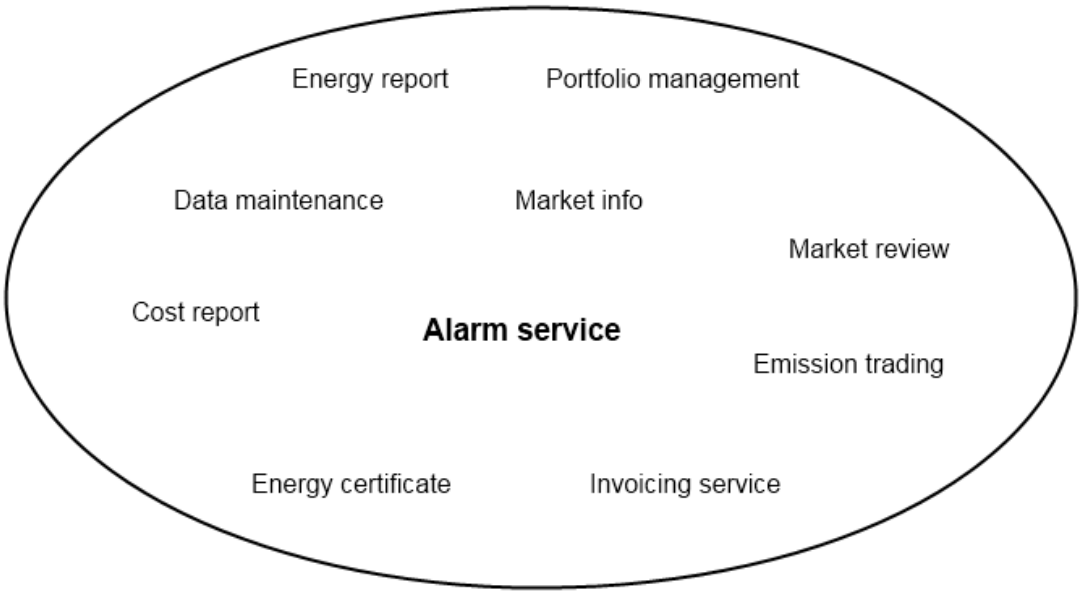

FIGURE 2: Basic overview of EnerKey's applications

Alarm is an energy consumption-threshold monitoring system for observing readings, which somehow exceed user-defined peaks, and informing the users about these peaks in a form of e-mail or some other electronic means of communication. As a part of EnerKey, Alarm is a supplementary service allowing its users to use energy more efficiently and eliminate unwanted expensive overruns in energy consumption. Alarm service is applicable on all energy types already known in EnerKey system because it is built as its extension. At the moment the solution encloses those energy types:

- Electricity
- Water
- Oil

- District Heating
- District Cooling
- Natural Gas

For each kind of energy there is a corresponding type of meter, actually there may be even more meter categories. Only certain alarms are allowed for each category of meters. Basically the user creates an alarm for a meter in one facility. The facility may contain many meters of many types. Figure 1 shows a simplified model of this architecture.

2 PROJECT REQUIREMENTS

In order to fulfill the product owner's expectations, a precise description of the system is necessary to be done before the development starts. This chapter declares requirements for Alarm. The word *Alarm* has two meanings in this thesis. First it is a name for the whole project and second it is understood as an **event that occurs when a sort of limit is overrun**, like a warning of a signaling device.

2.1 Alarm Types

The system is capable of creating following types of alarms ("Alarm descriptions.docx", 2011):

Cooling below threshold (week) Compares last full week's average cooling to a threshold value in Celsius. Alarm is triggered if cooling is less than threshold. Cooling indicates a difference in temperature. Alarm is set for a district heating energy meter.

Cooling below threshold (month) Compares last full month's average cooling to a threshold value in Celsius. Alarm is triggered if cooling is less than threshold. Cooling indicates a difference in temperature. Alarm is set for a district heating energy meter.

Consumption decrease (day) Compares last two sliding days' consumption to previous thirteen days' average, comparable consumption. Alarm is triggered if the two sliding days' consumption is less than specified relative decrease from previous thirteen days' average, comparable consumption. Only specified week days and hours are taken into account when calculating consumptions.

Consumption decrease (month) Compares last full month's average day consumption to previous month's average day consumption. Alarm is triggered if the month's average day consumption is less than specified relative decrease from previous month's average day consumption.

Consumption decrease (week) Compares last full week's consumption to previous three weeks' average consumption. Alarm is triggered if the week's consumption is greater than specified relative increase from previous three weeks' average consumption and is consumption difference is greater than specified threshold. Only specified week days and hours are taken into account when calculating consumptions.

Consumption decrease from last year (month) Compares last full month's average day consumption to previous year's corresponding month's average day consumption. Alarm is triggered if the month's average day consumption is less than previous year's corresponding month's average day consumption.

Consumption increase (day) Compares last two sliding days' consumption to previous thirteen days' average, comparable consumption. Alarm is triggered if the two sliding days' consumption is greater than specified relative increase from previous thirteen days' average, comparable consumption and if consumption difference is greater than specified threshold. Only specified week days and hours are taken into account when calculating consumptions.

Consumption increase (month) Compares last full month's average day consumption to previous month's average day consumption. Alarm is triggered if the month's consumption is greater than specified relative increase from previous month's consumption and the consumption increase is larger than specified threshold.

Consumption increase (week) Compares last full week's consumption to previous three weeks' average consumption. Alarm is triggered if the week's consumption is greater than specified relative increase from previous three weeks' average consumption and consumption difference is greater than specified threshold. Only specified week days and hours are taken into account when calculating consumptions.

Consumption increase from last year (month) Compares last full month's average day consumption to previous year's corresponding month's average day consumption. Alarm is triggered if the month's average day consumption is greater than specified relative increase from previous year's corresponding month's average day consumption and if average day consumption increase is larger than specified threshold.

Consumption over threshold (day) Compares last sliding day's consumption to a threshold value. Alarm is triggered is the sliding day's consumption is greater than specified threshold.

Consumption over threshold (month) Compares last full month's consumption to a threshold value. Alarm is triggered is the month's consumption is greater than specified threshold.

Consumption over threshold (week) Compares last full week's consumption to a threshold value. Alarm is triggered is the week's consumption is greater than specified threshold.

Hourly consumption maximum (day) Compares last sliding day's maximum hourly consumption value to the second largest value. Alarm is triggered if maximum value is greater than specified relative increase from the second largest value.

Hourly consumption maximum (month) Compares last full month's maximum hourly consumption value to the second largest value. Alarm is triggered if maximum value is greater than specified relative increase from the second largest value.

Hourly consumption maximum (week) Compares last full week's maximum hourly consumption value to the second largest value. Alarm is triggered if maximum value is greater than specified relative increase from the second largest value.

Hourly consumption maximum over threshold (day) Compares last full day's maximum hourly consumption to a threshold value. Alarm is triggered if maximum hourly consumption is greater than threshold.

Hourly consumption maximum over threshold (month) Compares last full month's maximum hourly consumption to a threshold value. Alarm is triggered if maximum hourly consumption is greater than threshold.

Hourly consumption maximum over threshold (week) Compares last full week's maximum hourly consumption to a threshold value. Alarm is triggered if maximum hourly consumption is greater than threshold.

Hourly consumption minimum over threshold (day) Compares last full day's minimum hourly consumption to a threshold. Alarm is triggered if minimum hourly consumption is greater than specified threshold.

Hourly consumption minimum over threshold (month) Compares last full month's minimum hourly consumption to a threshold. Alarm is triggered if minimum hourly consumption is greater than specified threshold.

Hourly consumption minimum over threshold (week) Compares last full week's minimum hourly consumption to a threshold. Alarm is triggered if minimum hourly consumption is greater than specified threshold.

Last reading Compares the age of the latest reading in hours to a threshold value. Alarm is triggered if the age in hours of the latest reading is greater than the specified threshold.

Missing readings (absolute) Compares the number of missing readings from specified time period to a specified threshold. Period selection specifies the resolution of the period. Period full specifies whether the period should be a full period or a sliding period. Period offset specifies how many selected periods are between the inspection time and the beginning of the time period. Period length specifies the length of the time period by the number of selected periods. Alarm is triggered if the number of missing readings is greater than specified threshold.

Missing readings (relative) Compares the relative number of missing readings from specified time period to a specified relative threshold. Period selection specifies the resolution of the period. Period full specifies whether the period should be last full period or a sliding period. Period offset specifies how many selected periods are between the inspection time and the beginning of the time period. Period length specifies the length of the time period by the number of selected periods. Alarm is triggered if the relative number of missing readings is greater than specified relative threshold.

Missing readings streak Compares the longest streak of missing readings from specified time period to a specified threshold. Period selection specifies the resolution of the period. Period full specifies whether the period should be a full period or a sliding period. Period offset specifies how many selected periods are between the inspection time and the beginning of the time period. Period length specifies the length of the time period by the number of selected periods. Alarm is triggered if the longest streak of missing readings is greater than specified threshold.

Reactive power over free portion Compares last full month's reactive power maximum hourly consumption value to electricity meter's last full month's maximum hourly consumption value's reactive portion. Alarm is triggered if last full month's reactive power maximum hourly consumption is greater than electricity meter's last full month's maximum hourly consumption's reactive portion and if the difference is greater than either specified free portion value or value defined in meter's facility's tariff. Alarm is set for an electricity meter.

Reactive power over threshold Compares last full month's reactive power's maximum hourly consumption value to a threshold value. Alarm is

triggered if reactive power maximum hourly consumption value is greater than specified threshold. Alarm is set for and electricity meter.

Reactive power portion over threshold Compares last full month's reactive power's maximum hourly consumption to last full month's electricity's maximum hourly consumption. Alarm is triggered if reactive power's maximum hourly consumption is greater than electricity's maximum hourly consumption multiplied with specified relative multiplier. Alarm is set for an electricity meter.

Temperature fixed consumption increase from last year (month) Compares last full month's temperature fixed consumption to previous year's corresponding month's temperature fixed consumption. Alarm is triggered if last full month's temperature fixed consumption is greater than specified relative increase from previous year's corresponding month's temperature fixed consumption and if consumption difference is greater than specified threshold.

Temperature fixed consumption decrease from last year (month) Compares last full month's temperature fixed consumption to previous year's corresponding month's temperature fixed consumption. Alarm is triggered if the month's temperature fixed consumption is less than specified relative decrease from previous year's corresponding month's temperature fixed consumption.

Temperature fixed consumption trend increase (month) Compares last full month's consumption's twelve-month-trend to previous full month's consumption's twelve-month-trend. Alarm is triggered if last full month's consumption's twelve-month-trend is greater than specified relative increase from previous full month's consumption's twelve-month-trend and if trend difference is greater than specified threshold.

Temperature fixed consumption trend decrease (month) Compares last full month's consumption's twelve-month-trend to previous full month's consumption's twelve-month-trend. Alarm is triggered if last full month's consumption's twelve-month-trend is less than specified relative decrease from previous full month's consumption's twelve-month-trend.

Warming below threshold (month) Compares last full month's average warming to a threshold in Celsius. Alarm is triggered if last full month's average warming is less than threshold. Warming indicates a difference in temperature. Alarm is set for district cold energy meter.

Warming below threshold (week) Compares last full week's average warming to a threshold in Celsius. Alarm is triggered if last full week's average warming is less than threshold. Warming indicates a difference in temperature. Alarm is set for district cold energy meter.

Zero readings (absolute) Compares the number of zero valued readings from specified time period to a specified threshold. Period selection specifies the resolution of the period. Period full specifies whether the period should be a full period or a sliding period. Period offset specifies how many selected periods are between the inspection time and the beginning of the time period. Period length specifies the length of the time period by the number of selected periods. Alarm is triggered if the number of zero valued readings is greater than specified threshold.

Zero readings (relative) Compares the relative number of zero valued readings from specified time period to a specified relative threshold. Period selection specifies the resolution of the period. Period full specifies whether the period should be last full period or a sliding period. Period offset specifies how many selected periods are between the inspection time and the beginning of the time period. Period length specifies the length of the time period by the number of selected periods. Alarm is triggered if the relative number of zero valued readings is greater than specified relative threshold.

Zero readings streak Compares the longest streak of zero valued readings from specified time period to a specified threshold. Period selection specifies the resolution of the period. Period full specifies whether the period should be a full period or a sliding period. Period offset specifies how many selected periods are between the inspection time and the beginning of the time period. Period length specifies the length of the time period by the number of selected periods. Alarm is triggered if the longest streak of zero valued readings is greater than specified threshold.

The product owner wants to have one common window for adding and modifying alarms.

2.2 Alarm Message

When the alarm is fired, a notification has to be sent to the user and its content has to be stored so the users of our system can open it and read it later. The message contains about 4 blocks displayed in tables. Examples can be seen in tables 1, 2, 3 and 4.

TABLE 1: Basic information about a facility

Facility id	945
Identifier 1	XYZ1
Identifier 2	-
Client	Company A
Facility's name	Facility A
Facility's address	Street 123
Postal area	Any
Meter's energy type	service water
Meter's name	Käyttövesi
Meter identifier	

TABLE 2: Time attributes of alarm

Inspection time	16.03.2011 09:00
Inspection period	14.03.2011 09:00 - 16.03.2011 08:59
Hours	01:00-04:59
Weekdays	Mon, Tue, Wed, Thu, Fri, Sat, Sun
Comparison period	01.03.2011 09:00 - 14.03.2011 08:59

TABLE 3: Values of alarm

Inspection period's consumption	2
Comparison period's average consumption	0,8
Difference	1,2

TABLE 4: Limits of alarm

Minimum difference	0
Increase percentage	30
Difference	1,2

Figure 3 is an example of an image that comes with the alarm message. In this figure can be observed that the user had defined a limit for hourly consumption maximum - green line. This value had been overrun twice, therefore an alarm was fired and a contact person was notified.

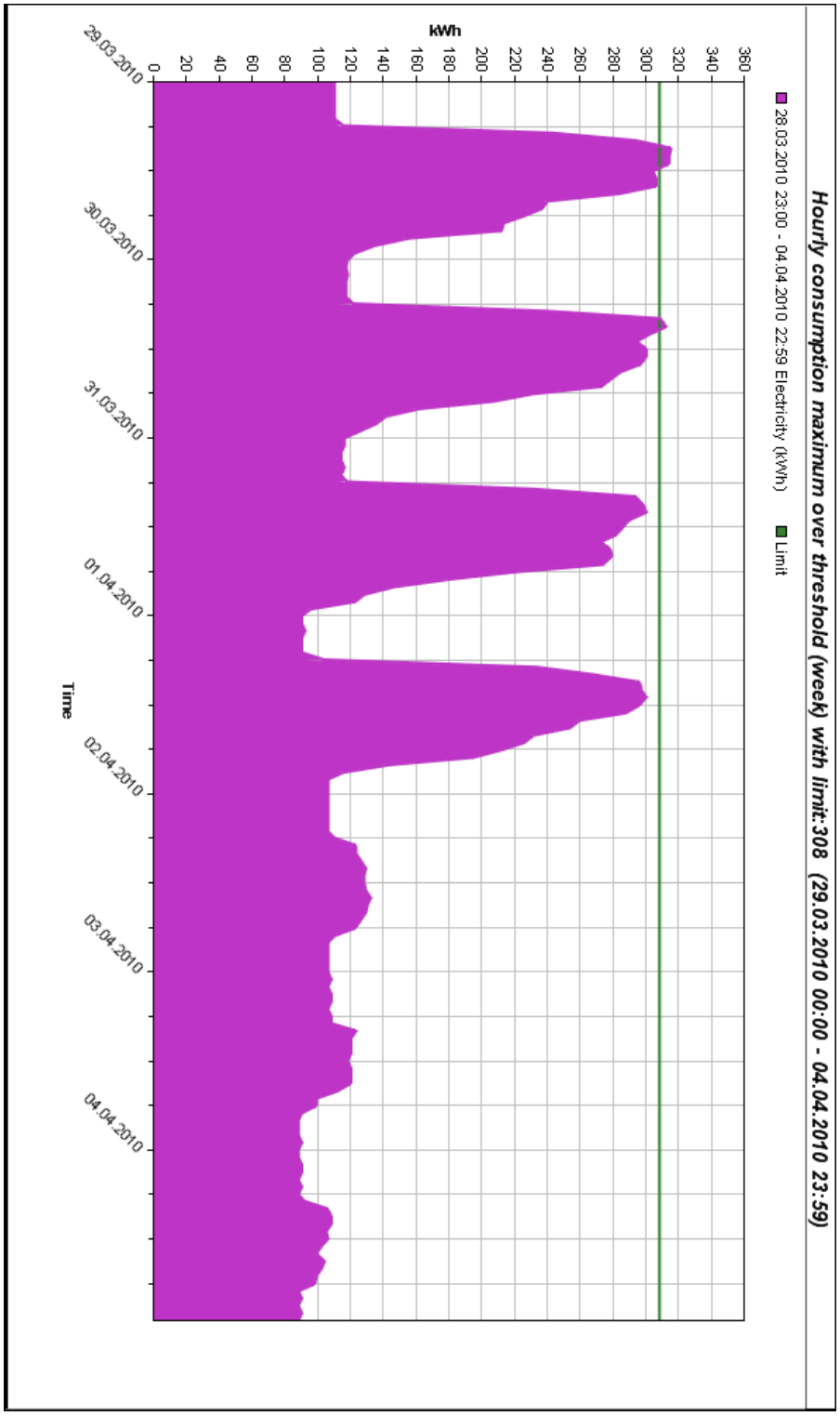

FIGURE 3: Image included in alarm report *(Energiakolmio, 2011)*

2.3 Alarm contacts

The messages generated after alarms go off are sent to customers' e-mail addresses. It is required that these e-mail messages go to people from these categories:

- Superintendent
- Maintenance
- Manager
- Owner
- EnerKey

In addition to these categories, also custom e-mail addresses must be allowed for the users. A special kind of contacts are supplementary service controls, for example XML export service. All these contact possibilities must be offered to the users when they create or modify alarms.

2.4 Functional Requirements

Alarm is supposed to become a part of EnerKey. Therefore it is required to support the same browsers and those are Mozilla Firefox, Internet Explorer and Google Chrome. Naturally, the user has to be authenticated and authorized to display only facilities of his own. When a facility is selected, the user is free to create many alarms for one meter as he might need to follow more information, being warned about various threshold overruns. What's more, the facility may (and usually does) contain more energy meters which need to be displayed in a sensible way, exactly informing the user about their placement. Lastly, the customer is interested in alarms created for his meters and wants a possibility to modify, delete and, in case of need, deactivate them.

Therefore the first part of user interface contains facilities (which is currently logged user authorized to see), their meters and alarms created for those meters. As there are thousands of facilities, meters and alarms in EnerKey, a filter, pager and a sorter are necessary to implement additionally. Moreover, commands like add, delete, export and modify are required to be a part of this UI as well.

A watchdog for monitoring consumption readings and firing alarms is a core requirement. In addition, the service has to be able to notify the customers via e-mail about this occurring event.

The second part of UI informs the users about executed alarms. The same information that was sent via e-mail should be available here, showing a text and image which describes the situation at the critical moment. Like in the first UI, sorting and paging are indispensable here.

Both user interfaces (pages) are accessible directly from the main menu of the EnerKey service, forming a new part of it and letting the user switch to another EnerKey application if needed. Therefore the menu from EnerKey has to be available from Alarm. In addition there was a request about resize ability of the pages. When the user changes the size of the window the inner content must be automatically rescaled but full readability of the table contents must be kept.

Because Energiakolmio is an international company and its products are offered abroad, more requirements are important for the success of this project. The system must have multi-language support so it can be translated into variety of languages. To utilize the time spent on this project as best as possible, the development team should focus on code re-usability and create common components that could be used somewhere else in EnerKey.

After considering functional requirements, first drafts of UI were made and presented to the development team. They will be discussed more in detail in the following chapters.

2.4.1 Page one

Our team was provided with figure 4. Due to fact that parts of the thesis are confidential the figure is omitted in the online publication. On the top of the page there is a table with header “Facilities and meters.“ In the very same container there are several buttons. The first one - Show filters - opens a window with filters to sort all facilities into groups the user wants to see and groups he/she does not, using criteria entered by him. Filtering options are based on properties of the filtered facility, e.g. name, energy type, meter type, company name. The same window serves as UI for filtering meters and alarms as well.

The purpose of the second button - “Executed alarms“ - is displaying page number 2 in a new window, accepting parameters from this user interface and therefore dynamically changing the content of the new window. The rest of the buttons implement data export functions, e.g. printing, writing into a “.pdf“ file and exporting into Microsoft Excel.

The table in the figure contains the most important information about facilities (name, address, company name), giving the user an opportunity to recognize them as effortlessly as possible. Those properties are an active part of the table header, carrying out the sorting function and making the navigation even faster. The first click sorts out the data in ascending order and the second click vice versa. In addition to this, the header contains a check box for selecting/deselecting all the facilities at once.

Further, each facility can be selected by clicking anywhere on the row or by hitting the check box. Deselecting is done in the same fashion. As mentioned above, a facility may contain more meters, and those meters may contain sub-meters and sub-sub-meters. No further hierarchy is allowed. By clicking on the arrow on the left the user can open the first level of the hierarchy, viewing the first level of meters.

A new sub-table is opened here, showing the basic data about meters in the facility like its name, energy type and meter type. All the columns are sortable in the same way as the main table. Even the selection is the same here. Finally, the last line of hierarchy has the same rules. These facts imply usage of a common component for all the tables.

Since there are hundreds of facilities in the company's database, paging has to be implemented as well. For this purpose there are arrows for moving to the first, previous, next and last page and also a direct page switcher. The user is allowed to set the page size as well and information about amount of pages and items is displayed. The pager will obviously be needed in all the tables, thus a common component would be handy again.

Selecting a facility makes the lower table called "Alarms" become active. It is filled up with alarms already made for the selected facility, facilities or meter. The structure and functionality is very similar to the upper table. The header contains a title and a button bar for controlling the table.

The first button adds a new alarm to selected meter, however it is inactive until a meter is selected. A click on this button opens up a window with a new alarm. Buttons for deleting and modifying an alarm become active when an alarm is selected, otherwise they stay gray, telling the user not to try to use them. This applies to all the buttons in this solution. Again, there is a button for filtering here, implying one more common component. In the end of the line there is an "active/passive" button, letting the user choose between enabled and disabled state of the alarm. When the alarm is disabled, it stays in the database; however the trigger does not fire any alarm. Using this button

for the second time makes the alarm active again, changing the command image as well.

The rest of the table lists through the created alarms, showing crucial information about them (name, active or passive state, timing interval and etc.)

The whole page is using AJAX so no reloads are necessary. When the user clicks either of the tables, an icon in the status bar next to the pager is activated and it starts indicating the loading process. It disappears after the table is ready. As both tables in this page are rather wide it is required to highlight the row where the mouse cursor is pointing at the moment.

At this point an experienced developer might notice a lot of functionality for a single page, demanding a sophisticated design pattern that would also allow satisfying maintainability and good possibility of manipulation with behavior of the page.

2.4.2 Page 2

The second part of the user interface contains already executed alarms since there is need for analyzing the fired alarms ex post, not only in the e-mail sent immediately after the alarm had been set off. As can be seen in figure 5 (due to fact that parts of the thesis are confidential the figure is omitted in the online publication), this interface contains particular instances of executed alarms, message blocks describing the event and a chart showing the measured reading overrunning the user-defined limit.

The table contains basic information about executed alarms. When the page is opened the table is sorted by date-time information of the executed alarms in descending order that the user sees the latest event first. Again, the headers of the table are active and allow sorting in both ways. When a concrete executed alarm is selected, a message and an image is displayed below the chosen row. When the user clicks on another row, the old row is collapsed and the new row is expanded. A pager is needed here as well since the number of executed alarms will raise in time.

2.5 Non-functional requirements

Even with a significant amount of functionality, a fast solution with a good user response and as low resource demands as possible is wanted. This has to be achieved despite the fact that the application must be fully compatible with the old EnerKey to become a part of it. At the same time it needs to be built with edge-cutting technology to enable effective maintenance and easy incorporation into new EnerKey in the future.

3 SYSTEM DESIGN

The objective of this chapter is to show what is going on "behind the scenes" of Alarm. The whole system is divided into two parts: ASP.NET web interface and alert service program. In the web interface, the user can set targets / indicators of consumption for the thresholds which cause alarms, go through executed alarms and analyze them etc. (see chapter 2.3 for all the functions). Alert service program checks the timer, fires alarms if necessary and sends message containing information about the event by e-mail to a predetermined address(es).

The focus of this thesis is on the web interface, which is built with multi-layer architecture, following the most up to date application development practices. The design of the web interface is described in this chapter, starting with persistence model, followed by middle layers and ending with introduction to a third party tool which was used to ease building the UI.

3.1 Persistence model

The goal of this sub-chapter is not to describe the whole database model, but only those parts essential for the last sprints of Alarm where the UI was developed. Figure 4 presents these parts in a class diagram as they were mapped with an ORM tool.

There are types of alarms in table *Alarm* in the database. These are basically just templates. The most important table is *AlarmInstance* which persists the actual alarms created by users. The relation between these two tables is one-to-many. One alarm has many *ConstantVariables*, *AlarmInstanceContacts*, *HourlyTimings*, *DailyTimings* etc.

Another substantial table is *ExecutedAlarmInstance* which archives concrete fired instances of alarms. In order to fulfill requirement about storing the message and the image of an alarm, tables named *AlarmInstancePicture* and *Message* were designed for this purpose.

The latest Entity Framework was used as the ORM, taking care of the difficult work between the SQL Server and the .NET application. EF produced the desired objects, however, most of these classes were not used as models for the presentation layer due to lack of some necessary properties on one hand and carrying too much information on the other hand. More suitable models were created in Energiakolmio's base libraries for the purpose of presenting. The Entity Framework object model can be observed in figure 4.

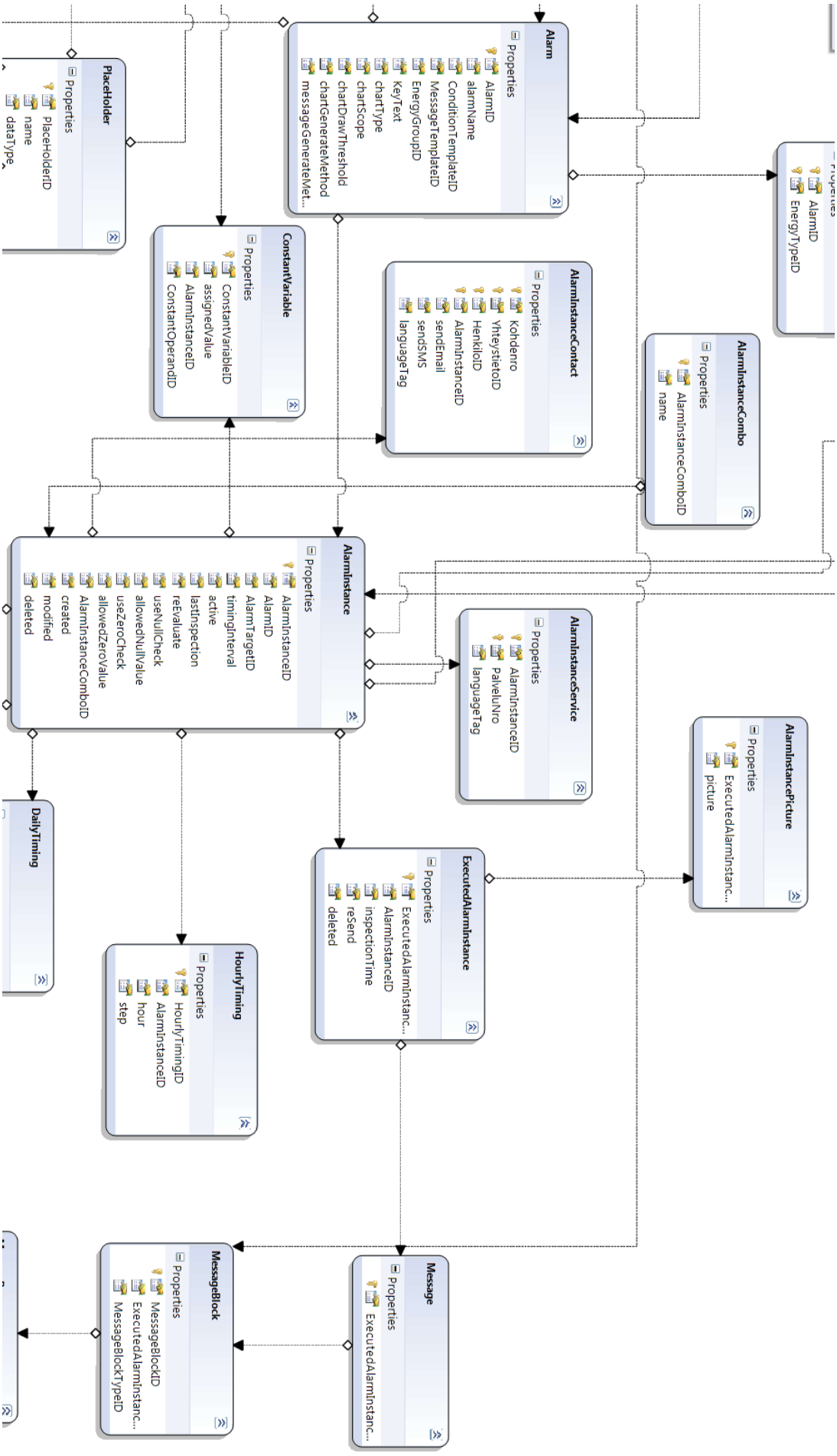

FIGURE 4: A part of object model *(Energiakolmio, 2011)*

3.2 Model-View-Presenter

To implement such a quantity of functionality in one page, the MVP software design was chosen for this project. It defines data flows and interaction between modules of the page. Using this architecture promises a little more effort when the team develops the project but the gain is significant when the product enters its maintenance life cycle phase. The other advantage of this architecture is better testability because the layers are strictly separated.

The ASP implements *IViewContainer* and is able to have many *ViewImplementations*, commonly known as user controls. In figure 5 there is a description of what a view implements. Each user control realizes one *IView* interface which prescribes behavior of the ASP control and a presenter. But the functionality is implemented in a presenter which is held as a property of the user control. The view sets its state through setters and routes user's commands to its presenter. This idea separates presentation logic from business logic. An example of this can be observed for instance in sub-parts of chapter 5.

Figure 6 presents the idea of relations between a presenter and a controller. The presenter has an instance of a controller(s), but only for getting application state data. Instances of filter providers and data providers ale held here as well, hiding the persistence layer from any higher level application layers. The presenter plays the role of a "middle-man." The only connection between the view and the persistence layer is the model that is used as a carrier of data to be displayed.

Typically the controller implements many *StateGetters* and *StateSetters*, remembering what has the user done in previous steps. The idea behind this structure is that the flow of state information is forced. The Views only see the application controller as their specific *StateSetter* interfaces. The Presenters only see the application controller as their specific *StateGetter* interfaces.

There is a life cycle of a model in figure 7. Data is queried by *IDataProvider* and passed to the presenter. From there the view takes them and shows them to the user. Now the user makes some changes to the model, the views passes it back to the presenter and the data is stored by a *IDataPersister* from here. The data provider also implements interfaces for filtering and data providing. There is also a filter definition provider which generates available filters from meta data, doing a lot of work for the development team. There is no need for writing separate filtering mechanisms in upper layers.

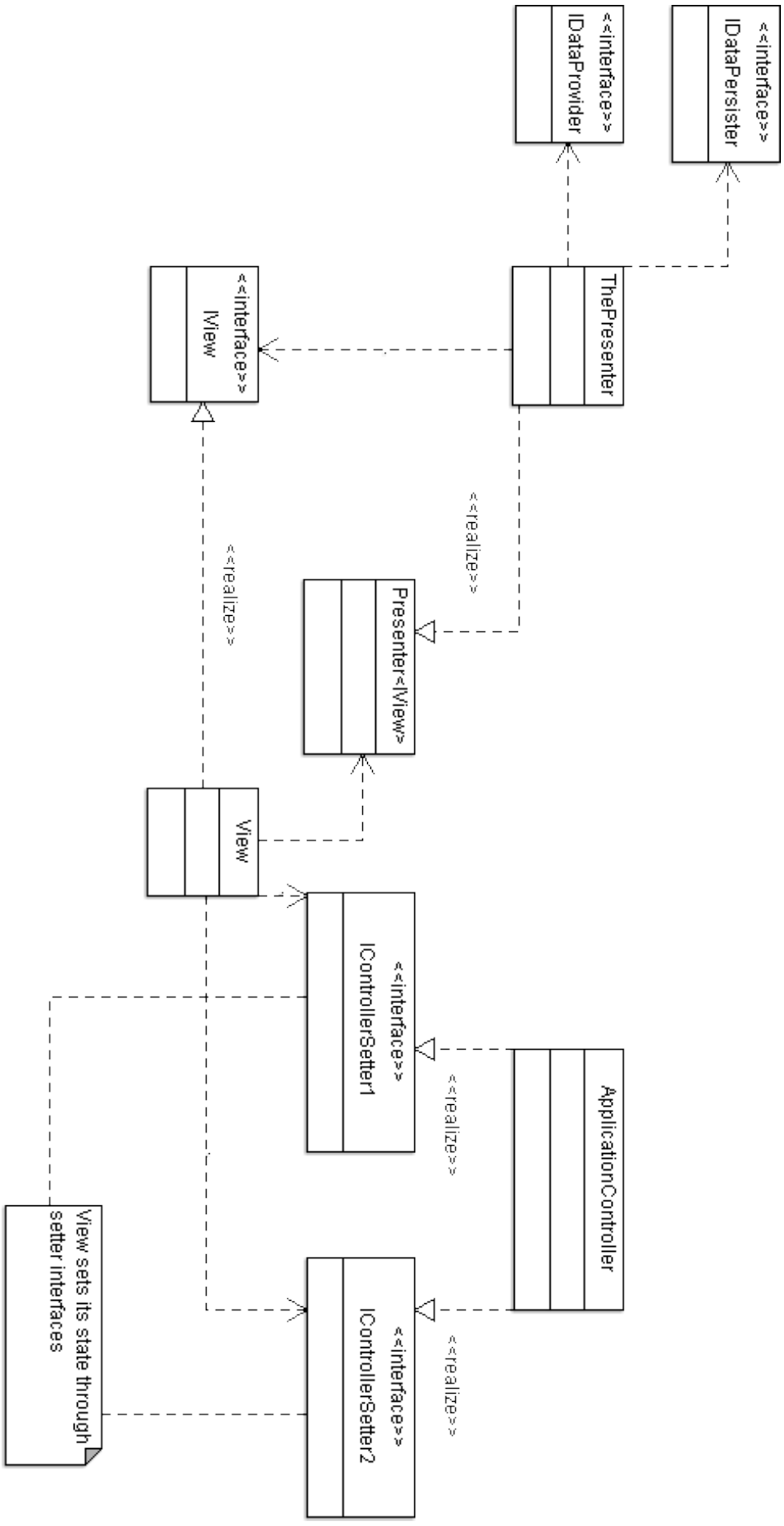

FIGURE 5: Relations between view and controller *(Energiakolmio, MVP, 2011)*

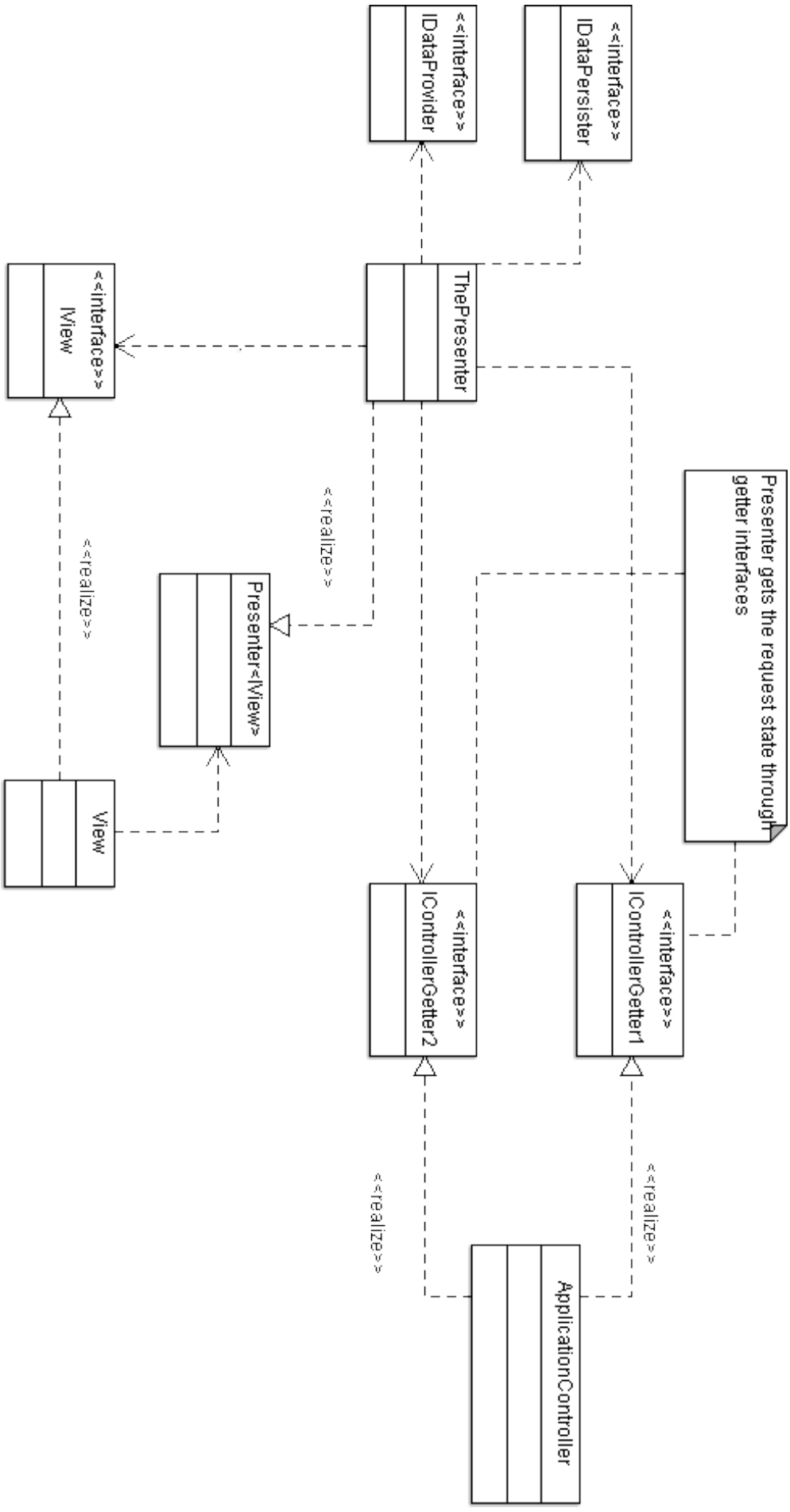

FIGURE 6: Relations between presenter and controller *(Energiakolmio, MVP, 2011)*

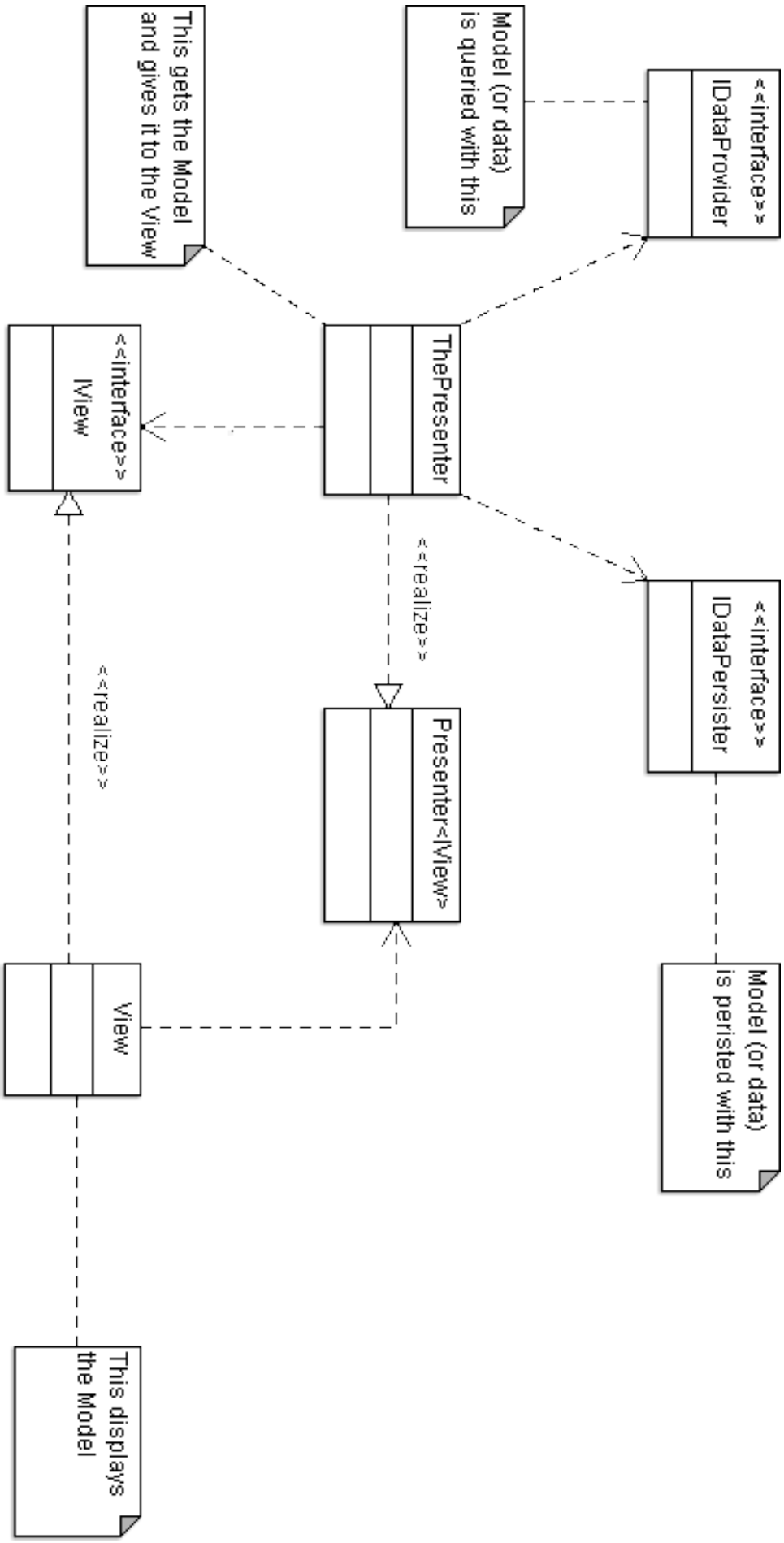

FIGURE 7: Life cycle of model *(Energiakolmio, MVP, 2011)*

3.3 Telerik ASP.NET AJAX Controls

Nowadays web developers have to implement common web page parts all over again in each new solution e.g. calendars, grids, tree views, windows, menus, site maps etc. Microsoft realizes this fact and provides ASP.NET with built in features, thus the common web page parts are available from the .NET libraries, making the developer's life much easier. As EnerKey is an enormous solution with many-sided requirements, the built-in features are not enough to cover them.

In addition, the first pages of the UI contain a lot of information, therefore reloading the page every time the user does any action would take unacceptable amount of time. This is where AJAX becomes useful as its purpose is to reload only necessary parts of the page, significantly influencing the user response time.

After considering both previous problems, Telerik's ASP.NET AJAX library was chosen to be used for Alarm. Its building blocks suit the requirements for this project, especially *RadGrid* with features like sorting, paging, scrolling, selecting, filtering, hierarchy, many column types, nested views and appearance adaptability. All these functions are needed, thus using Telerik saves time that would be necessary to develop them and lets the team solve project-specific problems. Another powerful block is *RadWindow* with its tabs and multi-pages. These controls are convenient for implementing functional requirements e.g. adding alarms, modifying alarms, delete confirmation and filtering.

4 DEVELOPMENT ENVIRONMENT

The development environment influences making software, thus it needs to be introduced here. Energiakolmio has been working with MS .NET platform and Alarm is not an exception. The platform contains applications and tools for creating software solutions. As the object of this thesis is a software project, a framework for project management was applied. This framework influences the implementation part as well, therefore a few words about it are mentioned in the sub-chapter 4.4.

4.1 Microsoft .NET 4 platform

The .NET Framework is a platform for building software of all kinds - console applications, web solutions, Windows forms applications and services. This is achievable because the framework applies common skills and programming tasks necessary to build to all these sorts of software. It is also integrated with other technologies and tools to build solutions with less work and faster.

The framework consists of these parts:

- CLR - provides an abstraction layer over the OS, allowing usage of multiple programming languages
- Base Class Libraries - pre-built code for common low-level programming tasks
- Development frameworks and technologies - solutions for larger programming tasks

The framework is very well documented in MSDN, therefore when there was an obstacle it was not a problem at all to find a solution for it. For development of Alarm C# programming language was used with its base class libraries and frameworks described in following chapters.

4.2 SQL Server Management Studio

As a database is crucial part of the whole system a tool is needed which would allow to manipulate with it. Alarm uses Microsoft SQL Server - relational database and analysis system for e-business, business and data warehouse solutions. SQL Server comes with so called SQL Server Management Studio - a tool for managing, administering all components and configuring the Server itself.

The Studio includes script editors for working with SQL directly and graphical tools as well to ease developer's work. Essential part of the Studio is Object Explorer which allows to browse through the server's objects like databases, security settings, replication settings, management settings etc.

For the author of this thesis the most important are Databases which contain Alarm's tables. From here the tables can be created, edited or deleted, and data can be manipulated as well. For doing this, knowledge of SQL is absolutely necessary.

After the database structure is created and some data is inserted, the developer is interested in response time of his queries. For measuring the performance the Studio contains execution plan tools which display potential bottlenecks in the queries and suggest creating indexes to speed up the process.

Another useful part of the Studio which the author worked with is SQL Profiler. From all of its functions, one was important for this project. It allows observing SQL queries from certain machines, users, applications. This function is very useful when a developer needs to watch a more complex application with many queries. The queries can be filtered out and finding information like the source of data is much easier.

4.3 Visual Studio 2010, Team Foundation Server

Microsoft Visual Studio is an IDE that supports entire application life cycle, from design to deployment. It is an ultimate all-in-one solution for developing applications for SharePoint, the web, Windows, Windows Phone.

Very important part for developing Alarm is source code control which is a part of Visual Studio. The project team shares the code via Team Foundation Server which is an upgrade from former ways of doing this (CVS, SVN). The Team explorer window offers a convenient way of updating and checking in the code, complemented by an opportunity to compare the current version of the file with historical versions from the Team Foundation Server. In case of conflicts there is a function called "auto merge," which speeds up solving this kind of problems. If auto merge cannot be used the Visual Studio has a tool that allows the user to solve the problem by hand in a comfortable way. In brief, Visual Studio proved to be a powerful instrument in this area.

Another strong property of VS is the Debug mode. Not only is it possible to put breakpoints almost anywhere the developer needs but it is also capable of opening files from other VS instances. It makes the debugging process

transparent and lets the developer focus on the real problem instead of trying to get out of a swirl of windows, files and variables and getting lost eventually. What is more, VS comes with a small development server which starts automatically when the debug mode is activated. This grants a chance to avoid painful installations of huge web servers and setting them up on all the workstations.

Last but not least, the code editor has to be mentioned here. VS emphasizes keywords of selected programming language enough to let the developer know about them but not too aggressively to make his or her eyes suffer. In addition, so called "intelisense" is sophisticated and easily accessible, speeding up the coding. Combined with refactoring options, multi-monitor support and possibility of using keyboard shortcuts, writing the source code of Alarm was effective.

4.4 Scrum

Generally, in the whole software industry, before Scrum was taken into practice, many waterfall software projects failed because they were not developed iteratively and the customers were put away, not seeing what is actually being made for them. In a case like this, the customers were shown what they paid for and it was something that was completely different from their expectations. In other industry branches this works diversely.

The products are being checked regularly and in the end there is a prototype that is tested. Then it is decided if the prototype already fulfills the expectations. If not, another development iteration is begun, improving the product continuously. This is exactly what the Scrum approach brings into the software industry.

As Scrum has proved to be an effective agile development method, Energiakolmio decided to implement it as well and it was used to develop Alarm. Starting in sprint 13, it was the very first Scrum project in the company. As with any other Scrum project, a product backlog was composed after considering requirements in chapter 2. Then smaller parts (tasks) of the backlog were developed in iterations as it can be seen in figure 8. Many words have been written about Scrum already, thus the author of this thesis wants to present its benefits that were particularly visible during the implementation of Alarm.

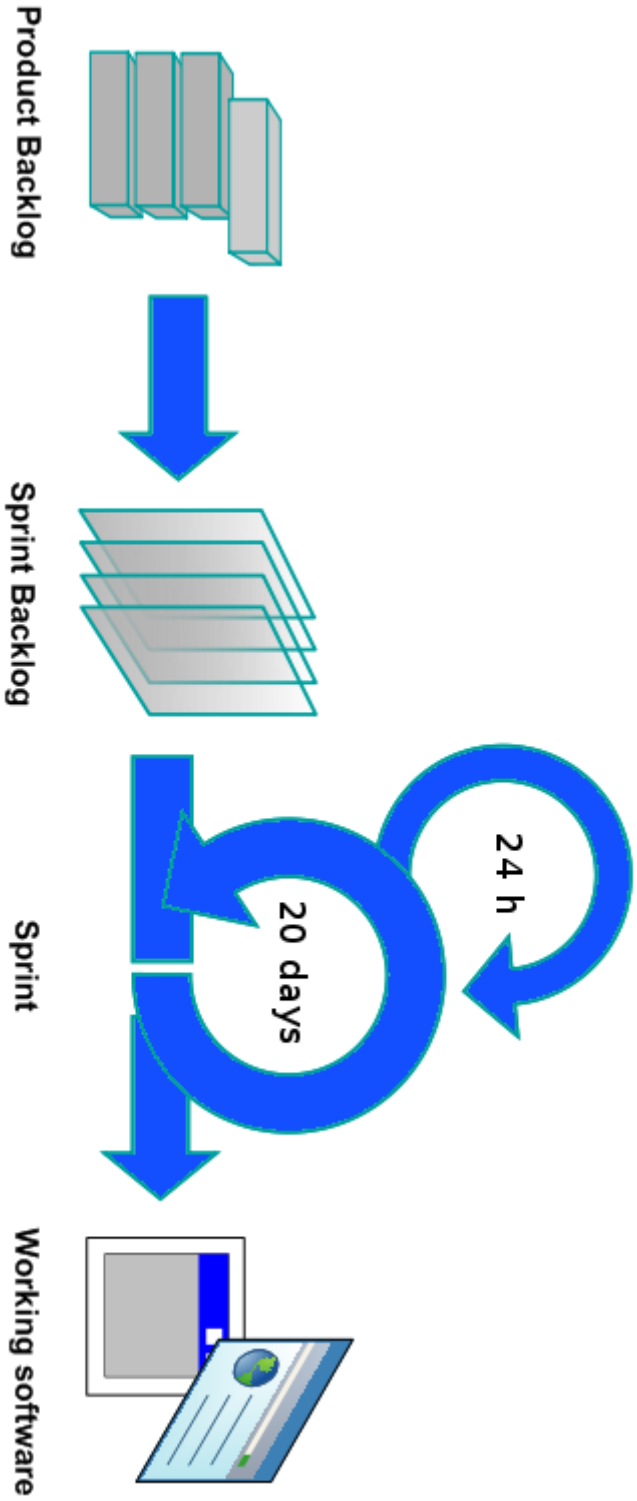

FIGURE 8: Scrum process *(Energiakolmio, Scrum E3, 2009)*

4.4.1 Realistic backlog

The first time in Scrum when the team gets together is the *Sprint planning meeting*. The main circumstances of the project are pointed out here by the product owner. User stories are presented and the team should get a clear idea what is going to happen from the product owner's presentation. It is not only the product owner's decision which stories will be finished in the upcoming sprint. This is negotiated and the team makes the decision to what they commit, **denying any unrealistic expectations** from the product owner in the very beginning.

All the stories are cut into smaller tasks and the team estimates the time needed for them. All the tasks are estimated separately and all the developers have to tell their estimates regardless their experience. Not to influence each other, the team members should present their estimates at the same time (this can be easily achieved by writing the estimates down on a paper, using "Planning poker" etc.) If the estimates for the tasks are too versatile, the task is discussed once again until it is clear for everybody. Then an average estimate is recorded into the product backlog.

The time for the project is limited by the team's total available working hours in the sprint. When the estimations have been done, the total estimated hours are compared to the available hours of the team. Since the amount of time is capped, the only way how to deal with a situation where there is not enough time, is to change the scale of the product. Simply - when there is too much work and not enough time, the team decides to leave out some tasks, making its commitment reasonable.

In the end of the sprint planning meeting, everyone knows what the goal is and that it is **realistic**.

4.4.2 Task track

During the development the team has to attend a *daily Scrum meeting* and present the work they have done. The Scrum master (the person in the Scrum team responsible for removing obstacles if they appear) can easily check on the progress of his/her project. Because of this, a precise **track of what is done** and what is left is kept. If there are any impediments outside the project, this is the time for the Scrum master to remove them, keeping the goal of the project achievable.

The individual team members are aware of their strengths and weaknesses certainly better than a project manager who divides the work in e.g. waterfall. Therefore it is an advantage of Scrum that the **team divides the work**. This also brings the team a significant amount of responsibility and commitments to each other.

In the Scrum the product owner (stakeholder, person who is investing in the Scrum project) is close to the Scrum team (developers) and gets an **early feedback** from his project, focusing on the customer value.

5 IMPLEMENTATION

The purpose of this part is to demonstrate how the project requirements from the chapter 2 were combined with with elements of the chapter 3 into a real web solution. At first a basic implementation of MVP in real application is shown in sub-chapter 5.1 and then solutions for non-trivial problems are presented.

The development team consisted of 4 developers and a Scrum master. Total amount of 442 hours during 2 sprints had been allocated for this project when the author of the thesis joined the group. The goal was to make the above described user interface and necessary underlying tiers, following the work that had been done before.

5.1 Deleting a single alarm

For this task the team needed a user control implementing an interface which describes this view and extending *TranslatableUserControl* (see chapter 5.4). The benefit of a user control is that it is independent on its placement in the page, keeping the functionality even when major changes are done to the page later. The describing interface *IDeleteAlarmCommandView* contains a public property indicating two possible Boolean states of the control: enabled and disabled. The source code of this simple interface can be seen in listing 1.

This user control has a private field of type *HtmlGenericControl* for representing the button itself, a presenter for this type of view, an instance of *IDeleteAlarmCommandStateSetter* for indicating user's confirmation and a translation key for a text description of the button.

Public attributes of the control are:

public string ClientClick - for setting client-side javascript confirmation function

public string PresenterKey - for setting the presenter

public string ControllerKey - for setting the controller

public string ExecuteClientID - for getting an ID of the button

public bool Enabled - for getting and setting property of the implemented interface

public event EventHandler Deleted - for populating the event of deleting an alarm

```
namespace EK.MVP.Alarm.View
{
    public interface IDeleteAlarmCommandView
    {
        bool Enabled { get; set; }
    }
}
```

LISTING 1: Interface IDeleteAlarmCommandView

In a protected method *Page_Init* a base method of the same name is called, the service locator is used to get a language getter, a presenter and a controller (depending on public attributes which are set in the markup.) The service locator is asked for these instances in this case. Additionally, the generic HTML control is instantiated and set to be an HTML input of image type and attributes *alt* and *title* are set here to display balloon tips when the mouse is over this button. Inherited *TranslatableUserControl* is conveniently used to do the translations.

In the end an event *Page.LoadComplete* is bound to a method called *Page_LoadComplete*. From there a method *OnViewInitialized* is called. Inside, the controller is asked about an ID of the selected alarm, leaving the field as a null if none is set and deleting the selected alarm in case the ID had been set and the user confirmed deleting with current post back.

The next phase of the page life cycle is the *PreRender* event. The *OnClick* javascript attribute is set here. Furthermore *OnViewLoaded* method of the presenter is called from here, trying to determine the availability of the button. For this purpose the presenter contains a data provider and an alarm instance getter which is asked for a selected alarm. The button should be enabled only if a selected instance of an alarm exists. Otherwise the user cannot delete anything of course. Then, back in the view, images for events *onmouseover* and *onmouseout* are assigned. Previously mentioned property *Enabled* is used here to determine which picture to use as the “inactive“ button must melt into the background.

Controls for modifying and switching alarm status are implemented in the same fashion. The only specialty is in *IActivePassiveCommandView* because this is a two-purpose user control. Its function can be defined in the markup by setting a Boolean switch to true if an activator is needed and false if deactivator is needed. In brief there are two buttons implemented in one view with a switch.

5.2 Selecting a single alarm from listing

It is the Telerik data grid from section 3.3 which proved to be useful for making the alarm listing. The option of custom sorting of the grid is used here as well as the built-in paging possibilities. *ItemDataBound* and other events of the grid are taken advantage of for translating incoming data, adding client-side functionality, selecting the right icons for the current states of alarms and responding to the user commands on the grid.

Interface in listing 2 enforces the view to implement some behavior. First of all there must be an alarm instances setter, then a row count setter is needed for keeping a virtual row count. In addition the user is able to set the currently displayed page and the page size, thus setters for these attributes are needed. The getters are implemented because of the filtering.

```
public interface ISetAlarmsListView
    {
     int? SelectedAlarmInstanceID
        {
            get;
            set;
        }

        int RowCount { set; }

        int PageNumber
        {
            get;
            set;
        }

        int PageSize
        {
            get;
            set;
        }
    }
```

LISTING 2: Interface ISetAlarmsListView

The view also populates events like *PagingChanged, SortingChanged, RowStateChanged* to inform other user controls about changes so that they can react. For example, controls for modifying and deleting need to be informed about alarm selection changes to switch their states from enabled to disabled and vice versa. There is an update panel which holds this view and the mentioned controls, therefore it is enough to refresh the panel and all the inner child controls are refreshed. Furthermore this view needs those attributes:

- private Presenter<ISetAlarmsListView> - presenter

- private IPagingSettingsSetter - for paging purposes
- private ISortingSettingsSetter - for sorting purposes
- private IVisiblePropertySetter - not all data in the data source are displayed
- List<Tuple<string, IEnumerable<IPropertyMetaData>>> - meta data information
- private ISelectedAlarmInstanceIDSetter - instance of a controller
- private IDeleteAlarmCommandStateSetter - sets a flag of an alarm for deleting

The presenter has a data provider and most of the getters that have the opposite function to these setters. For all the listed setters there is only one controller instance that stores all the information into the persistent application state.

5.3 Display executed alarm

The executed alarm can be selected in the page with executed alarms, where it is listed in another *RadGrid*. By clicking on a row the line is expanded and message blocks and an image are shown to the user. This is done by following two user controls.

5.3.1 Message of an alarm

The output of this user control is a dynamic number of the tables described in chapter 2.2. To have the dynamic count of blocks, two repeaters were used, one inside another. The problematic part of this was the fact that the inner repeater could not be accessed from the code behind, thus a data source could not be set there.

This problem had to be solved in the markup as can be seen in listing 3. After casting to an enumerable class, the message presentation object is set as a data source to the inner repeater. In order to the repeater be able to do this, interface *IEnumerable* had to be implemented in a way that an enumerator over message rows collection is returned from the class of the message presentation. Finally, the second repeater is able to access the rows and evaluate their content. The alternating template was used because of highlighting every other line.

```
<asp:Repeater ID="tableRepeater" runat="server">
    <HeaderTemplate>
    </HeaderTemplate>
    <ItemTemplate>
        <table cellpadding="0" cellspacing="0" border="0" class="alarm_description
            ">
            <tr>
                <th colspan="2">
                    <%#DataBinder.Eval(Container.DataItem,"HeaderText")%>
                </th>
            </tr>
            <asp:Repeater ID="childRepeater" runat="server" DataSource='<%# ((
                IEnumerable)Container.DataItem)%>'>
                <ItemTemplate>
                    <tr>
                        <td style="width: 50%;">
                            <%#DataBinder.Eval(Container.DataItem,"Label")%>
                        </td>
                        <td class="rightalign">
                            <%#DataBinder.Eval(Container.DataItem,"Value")%>
                            <%#DataBinder.Eval(Container.DataItem,"Unit")%>
                        </td>
                    </tr>
                </ItemTemplate>
                <AlternatingItemTemplate>
                    <tr>
                        <td>
                            <%#DataBinder.Eval(Container.DataItem,"Label")%>
                        </td>
                        <td class="rightalign">
                            <%#DataBinder.Eval(Container.DataItem,"Value")%>
                            <%#DataBinder.Eval(Container.DataItem,"Unit")%>
                        </td>
                    </tr>
                </AlternatingItemTemplate>
            </asp:Repeater>
        </table>
    </ItemTemplate>
</asp:Repeater>
```

LISTING 3: AlarmMessageView - markup

5.3.2 Image of an alarm

Telerik's binary image component is used for displaying this image. It is an extension of ASP image that only needs an array of bytes as the image source and the HTTP handler is done for the developer. Another advantage is that in the case the image is not available for any reason, this component is degraded back to a classic ASP image, allowing the developer to set a default image for situations like this.

5.4 Universal translation component

In addition to the mentioned user controls, there are 38 views for all kinds of alarms in the system. Their common property is a translator as it is required for the whole project to be multi-lingual. Implementing the translation mechanism separately in each component is unthinkable, therefore a universal user control was written and the views in Alarm extend this component.

```
public class TranslatableUserControl : System.Web.UI.UserControl
    {

        protected ITranslator  translator ;
        protected ILanguageGetter languageGetter;

        protected string Translate(string key)
        {
            string  result ;
            translator . TryTranslate(key,  languageGetter.LanguageTag, out result,
                true);
            return  result ;
        }

        protected string Translate(string key,  bool  capitalize )
        {
            string  result ;
            translator . TryTranslate(key,  languageGetter.LanguageTag, out result,
                capitalize);
            return  result ;
        }

        protected void Page_Init(object sender, EventArgs e)
        {
            translator  = ServiceLocator.Current.GetInstance<ITranslator>("
                AlarmTranslator");

        }
    }
```

LISTING 4: TranslatableUserControl

The code can be observed in listing 4. The translator is instantiated here because all the views should use the same translator for the translations and the language getter is set by an injection. The methods for translation do the actual work, knowing that the injected translator will have methods for translating. This is because the translator implements interface *ITranslator*. The injection is done in the *Page_Init* that is called in extending classes. These classes do not have to know what is the actual translator, thus they simply call *base.Page_Init(sender, e);*. The only thing that these classes have to implement is to set a controller as a language getter in the overriding *Page_Init* method and finally the *Translate* methods are ready to be used.

6 RELEASE VERSION

This chapter presents the final product as it was shown to the product owner. Alarm is not available on the web for the public as it is meant to be used by Energiakolmio and its customers. The English language mutation of screen shots was chosen for the purpose of this thesis. It also needs to be stated that the windows shown here are not pop-ups but they appear in the center of the page, dimming the rest of it when they appear.

6.1 Main pages

Unlike the page with the executed alarms which has stayed the same as it was designed, the first page went through some modifications. It was caused by changing visual requirements of the product owner. The final result can be seen in figure 11 (due to fact that parts of the thesis are confidential the figure is omitted in the online publication).

The search bar function from the UI proposal is done in the filter window, therefore it is not needed to have it on the page. The columns of the tables also had to be changed. Next, the number of command buttons was reduced, the pager was extended and the appearance of the page as a whole was improved.

6.2 Filter window

As can be observed in figure 12 (due to fact that parts of the thesis are confidential the figure is omitted in the online publication), there are three tabs in the header, allowing the user to set filters for facilities, meters and alarms in a one common window. The button for opening this window is located in two places on the first page (figure 4, blue buttons). When the user clicks, the related tab is opened for him/her based on the button that was used. This is done by a client javascript function. On each row there is an attribute which is used as a base for sorting, followed by an operator. The operator can be:

- Equals
- Is not equal to
- Is greater than
- Is lesser than
- Is greater or equal to
- Is lesser or equal to
- Is contained in

The variety of operators changes on the sortable type. On the right side of the operator there is usually a text input but on this picture there are some elements that can be chosen. Is also possible to apply more filters at the same time. For this purpose, button “Add“ is placed below the rows and button “Remove“ can be found in the end of each row. The free space below the row in the picture is taken when there are more filters applied. AJAX is used for most of this functions here, for example for returning the operators and the right side, adding new rows, applying filters etc. In the lower part of the window two buttons can be observed. One is for clearing and one for using the filters.

6.3 Add alarm wizard

The act of creating a new alarm consists of three steps that are subsequently done in a one wizard window. The first of them can be observed in figure 13. It is a significant drop-down list with all the types of alarms from section 2.1. The user simply picks one and continues with the “Next“ button.

The next step can be watched in figure 14 (due to fact that parts of he thesis are confidential the figure is omitted in the online publication). Based on the selected facility the people from groups in chapter 2.3 can be marked and unmarked as receivers of the alarm messages. In addition the language of the message can be selected here as well. Below there is a possibility to enter custom e-mail contacts if the Energiakolmio’s internal contact database is not sufficient as a source. In the last row there is the option to select a special XML message export. The user moves on or back with the buttons “Previous“ and “Next“.

The content of the last window in figure 13 is generated, based on the selected type of alarm. Considering the count of all the alarm types, the author of this thesis does not intend to present all possible generated contents. In this most common step the user enters alarm variables, quality settings and timing settings. The last check-box is for setting the state of the alarm to active or passive (green and red icons in figure **??**). The process of creation is either ended by the button “Save“ or the user can go back with the button “Previous“. When using the buttons for moving between steps in the wizard, the already filled data is saved in the *view state* of the page, keeping the information for the user.

There is also a “modify alarm wizard“ that uses the same windows and steps, thus it is not shown in this place. The only difference is that the first step is not present because once the alarm is created, its type must not be changed.

6.4 Miscellaneous details

A confirmation for deleting an alarm is needed to prevent the user from accidentally deleting an alarm. Another Telerik RadWindow control is used for this purpose as can be seen in figure 16 (due to fact that parts of the thesis are confidential the figure is omitted in the online publication).

In the figure 17 (due to fact that parts of the thesis are confidential the figure is omitted in the online publication) it can be observed how the paging is done in the project. It is a part of Telerik's *RadGrid* that can be completely translated and so it is. When the mouse is over the buttons, bubble tips are shown to the user and the page size is customizable as well. Considering the performance of the system, a count of 250 was chosen to be the maximum allowed number of facilities and alarms per page. Finally on the left side there is a square where a loading icon is displayed when the user enters a command and waits for the AJAX response.

There is a detail of the right part of the alarm listing in figure 17 (due to fact that parts of the thesis are confidential the figure is omitted in the online publication). For the user's comfort the order of the buttons was changed compared to the UI proposal and also the content is slightly different. The selected alarm is highlighted and the same icons as for activating and deactivating alarm are used here. Moreover, the pager was extended so that it shows more detailed information about current page in the list and it has a total count as well.

7 PROJECT EVALUATION

The last chapter presents the results of the project from all points of views - the customer, the management, the author of this thesis and the company as a whole.

7.1 Achievements

Because Alarm is a system made for a company, the achievements are mostly practical. The main goal of the last sprints was to prepare Alarm for going to production and this was achieved even though some minor changes before the release are being considered now. From the company's point of view, the development team performed very well and will continue on more projects.

In addition, re-usable components were built and they can be used in other EnerKey projects from now on.

Alarm produces a valid HTML code and the used MVP architecture and Telerik components proved their points and will be used in new projects as well. It is important to have a verified architecture for Enerkey since there is a plan to rebuild some parts in the near future.

Furthermore, Energiakolmio's Scrum planning can be improved based on the team's performance, particularly the backlog creation and sprint planning meeting as those were mentioned during the retrospective Scrum meeting when the project had finished. In addition, this thesis itself can serve as a source of knowledge when the company's new systems are developed.

7.2 Testing and feedback

After the development had finished, a testing environment was established. The first release version of Alarm was published there and the product owner and his team were asked for testing the system and feedback. This is a statement from an interview between author of this thesis and the product owner , who speaks for his team:

> *Alarm service work estimation for sprints 13/18 and 14/18 was 442 hours. The goal was to get the first release of Alarm to production. The team's task was to create a user interface with these functions: . . .* (the requirements were already written in section 2.3) *The test version revealed additional needs in user interface, thus the workload increased*

> *from the estimate. For this reason the product owner should have access to the test version in advance even when the user interface is unfinished yet. We already have improvement ideas for the team. After a few improvements the product will be ready to go to production. Furthermore, we have tested all the functions from the product backlog and the functionality meets our needs. We were especially satisfied with the layout of the user interface and the development team that was working and communicating well together.* (Kokkonen, 2011)

The project manager was asked about his evaluation as well. He stated this:

> *The Alarm project was a big scale project in our environment and also the first Scrum project in our company. In the first twelve sprints the work was not visible for product owners and that is one thing we have to change in the future. We also have to keep in mind that Alarm project was our first Scrum project. If we compare the results to that, I think we have managed very fine. The team has learned a lot during the project and especially in the last sprints the work was very efficient. The communication between team and product owner has developed a lot and the learning goes on.* (Hiltunen, 2011)

7.3 Future improvements

When the development had finished the team and the product owner had some ideas for future improvements of Alarm. The first of them could be making the green/red images in the alarms listing table active, allowing the user to change the state of the alarm directly in the row and avoid using the button. Another upgrade could be a multiple alarm selection to enable the users to delete and activate/deactivate more alarms at the same time with only one mouse click.

On the second page it could be allowed to display more executed alarms at the same time instead of collapsing the alarm every time another alarm is selected.

7.4 Conclusion

By the time I am writing these lines a last little Scrum before deployment is being prepared even though people in our ICT team agree that Alarm is now the best part of the whole EnerKey. This fact proves that software must be implemented in iterations and needs to be checked regularly as it is developed in order to meet the requirements of real users.

My personal experience with this project was positive as I had the chance to use the skills I learned during my studies and at former work placements in my home country. It was also my first development project where English was the only language and it was not a problem to exchange information with others. I was also officially reassigned to the team I developed Alarm with, as before we had started I was in another group. What is more, I was offered to stay in Energiakolmio even after my practical training is finished.

Thinking about my professional competences, I have improved my overall skills with Microsoft .NET platform, like working with Visual Studio and SQL Server Management Studio and finally - I have learned a new web architecture, which will be useful in my future career. Using ASP was useful to me as well because earlier, besides school projects, I did not have any significant experience with it. Considering my negative attitude to JavaScript, this has improved as well.

When we were asked about negative matters during the last sprint, I did not hesitate to tell my opinion about the user interface proposal being delivered too late. I think that there has to be a constant improvement in order to keep software development up-to-date and effective. Now I am looking forward to more work with my team as developing Alarm was quite enjoyable.

REFERENCES

Kokkonen, A. 2011. Product Manager, Energy Efficiency Services, Energiakolmio, Interview of 4th of April 2011.

Hiltunen, J. 2011. Project Manager, ICT, Energiakolmio. Interview of 26th of April 2011.

Energiakolmio's web presentation, 2011, `http://www.energiakolmio.fi`

Energiakolmio, 2011, "Alarm descriptions.docx"

Energiakolmio, 2011, MVP

Energiakolmio, 2009, Scrum E3